Razvoj metode za steganografiju i stegoanalizu u računalnim slikama

Kunštek, Toni

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:295879

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-11-24**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

RAZVOJ METODE ZA STEGANOGRAFIJU I STEGOANALIZU U RAČUNALNIM SLIKAMA

Diplomski rad

Toni Kunštek

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 13.09.2023.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Toni Kunštek
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1215R, 08.10.2021.
OIB studenta:	60222666607
Mentor:	prof. dr. sc. Irena Galić
Sumentor:	Marin Benčević, mag. ing. comp.
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	doc. dr. sc. Krešimir Romić
Član Povjerenstva 1:	prof. dr. sc. Irena Galić
Član Povjerenstva 2:	dr. sc. Marija Habijan
Naslov diplomskog rada:	Razvoj metode za steganografiju i stegoanalizu u računalnim slikama
Znanstvena grana diplomskog rada:	**Obradba informacija (zn. polje računarstvo)**
Zadatak diplomskog rada:	Proučiti i opisati metode steganografije na računalnim slikama kao i metode detekcije steganografije (stegoanalize) koje se temelje na metodama obrade slike. Razviti i opisati vlastitu metodu steganografije. Ispitati metodu na primjerima pod različitim uvjetima veličine i tipa skrivene poruke. Opisati prednosti, nedostatke i ograničenja metode te usporediti s postojećim metodama. Razviti i opisati metodu detekcije razvijene metode steganografije. Ispitati kvalitetu metode detekcije pod različitim uvjetima i statistički obraditi rezultate. Usporediti rezultate s drugim poznatim metodama stegoanalize. Sumentor: Marin Benčević. Rezervirano za: Toni Kunštek.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 2 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina

Datum prijedloga ocjene od strane mentora:	13.09.2023.
Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 25.09.2023.

Ime i prezime studenta:	Toni Kunštek
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1215R, 08.10.2021.
Turnitin podudaranje [%]:	3

Ovom izjavom izjavljujem da je rad pod nazivom**: Razvoj metode za steganografiju i stegoanalizu u računalnim slikama**

izrađen pod vodstvom mentora prof. dr. sc. Irena Galić

i sumentora Marin Benčević, mag. ing. comp.

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1
   1.1. Zadatak diplomskog rada 1
2. PREGLED PODRUČJA 2
   2.1. Steganografija 2
   2.2. Steganaliza 4
3. OPIS METODA 5
   3.1. Sekvencijalna LSB metoda 5
   3.2. Patch LSB metoda 6
   3.3. Dekodiranje 7
   3.4. Metoda steganalize 10
      3.4.1. Chi-square test 10
4. IMPLEMENTACIJA RJEŠENJA 12
   4.1. Programski jezik python 12
   4.2. Implementacija sekvencijalne LSB metode 13
   4.3. Implementacija patch LSB metode 15
   4.4. Implementacija postupka dekodiranja 19
   4.5. Implementacija metode steganalize 22
5. REZULTATI I USPOREDBA STEGANOGRAFSKIH METODA 23
   5.1. Rezultati sekvencijalne LSB metode 23
   5.2. Rezultati patch LSB metode 28
   5.3. Usporedba sekvencijalne i patch LSB metode 29
      5.3.1. Usporedba po količini spremanja podataka 30
6. ZAKLJUČAK 32

LITERATURA 33

SAŽETAK 35

ABSTRACT 36

ŽIVOTOPIS 37

PRILOZI 38

1. UVOD

Razvoj interneta i široka uporaba raznih komunikacijskih digitalnih tehnologija značajno su povećali količinu i dostupnost informacija čime su nastali novi izazovi u zaštiti podataka i osjetljivih informacija. Neovlašteni pristup informacijama, povrede podataka i hakiranje postaju sve veće prijetnje koje navode pojedince, organizacije i vlade da zaštite svoje podatke i osjetljive informacije. Postoje različite strategije za osiguravanje povjerljivosti podataka, a jedan od tih načina je steganografija. Steganografija uključuje skrivanje tajnih informacija unutar medija kao što su slike, tekst i audio datoteke. Steganografija potječe iz davnih vremena, ali se najviše razvila u digitalnoj eri pomoću digitalnih datoteka kao nosioca skrivenih poruka. Postupak kojemu je cilj razotkriti skrivene informacije naziva se steganaliza. Ovaj postupak koristi različite analitičke tehnike za otkrivanje skrivenih poruka. Stalni napredak u oba polja uzrokuje stalnu utrku u naoružanju između prikrivanja i otkrivanja, a time i potrebu za istraživanjem i inovacijama. Razumijevanje zamršenosti steganografije i steganalize ključno je za razumijevanje implikacija i potencijalnih protumjera povezanih s tehnikama tajne komunikacije u našem, međusobno povezanom digitalnom svijetu.

U poglavlju „PREGLED PODRUČJA“ opisane su različite metode steganografije. Zatim su u poglavlju „IMPLEMENTACIJA RJEŠENJA“ opisana programska rješenja dviju vrsta steganografija i postupak steganalize u programskom jeziku python. U poglavlju „USPOREDBA STEGANOGRAFSKIH METODA“ uspoređene su učinkovitosti dviju implementiranih metoda steganografije. Naposljetku se nalazi rezimirajuće poglavlje „ZAKLJUČAK“.

1.1. Zadatak diplomskog rada

Proučiti i opisati metode steganografije na računalnim slikama kao i metode detekcije steganografije (stegoanalize) koje se temelje na metodama obrade slike. Razviti i opisati vlastitu metodu steganografije. Ispitati metodu na primjerima pod različitim uvjetima veličine i tipa skrivene poruke. Opisati prednosti, nedostatke i ograničenja metode te usporediti s postojećim metodama. Razviti i opisati metodu detekcije razvijene metode steganografije. Ispitati kvalitetu metode detekcije pod različitim uvjetima i statistički obraditi rezultate. Usporediti rezultate s drugim poznatim metodama stegoanalize.

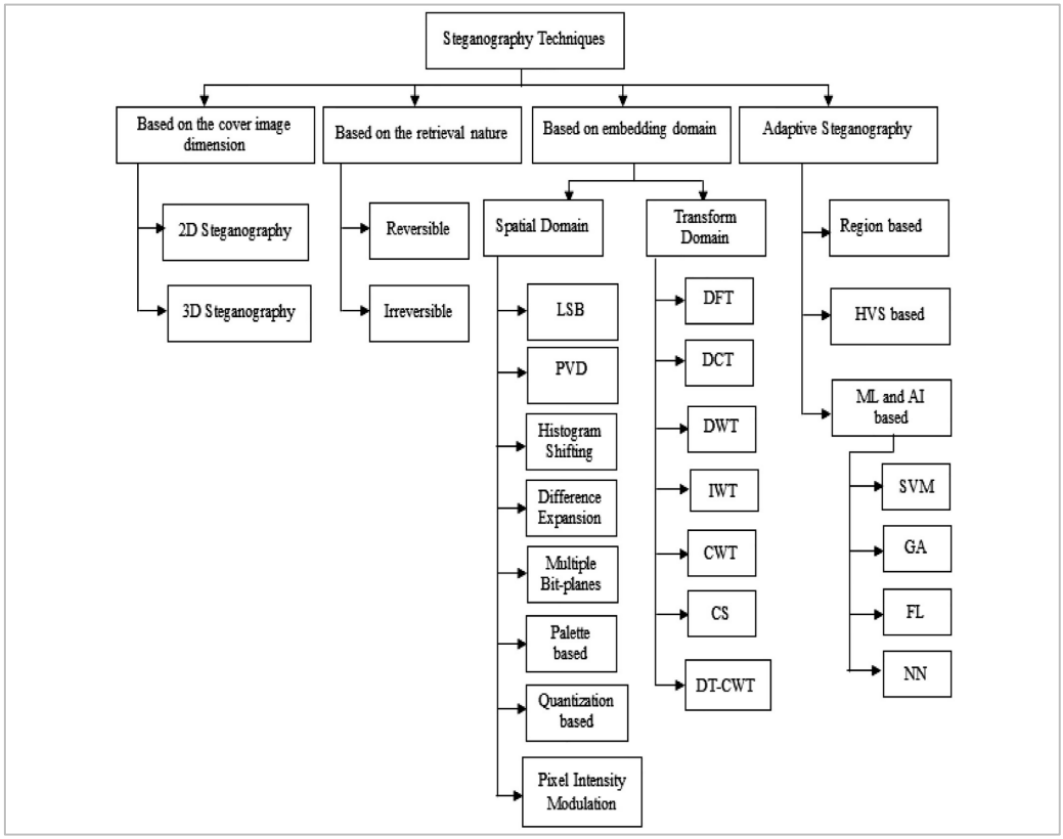
2. PREGLED PODRUČJA

Steganografija se odnosi na znanost o „nevidljivoj“ komunikaciji. Za razliku od kriptografije, gdje je cilj osigurati komunikaciju od prisluškivača, steganografske tehnike nastoje sakriti samu prisutnost poruke od promatrača. [1]

Osnovna ideja iza digitalnog steganografskog procesa je sakriti tajne ili privatne informacije unutar medija na način koji se ne može otkriti. Vrsta tajnih podataka može varirati od binarnih bitova, tekstualnih podataka, slika i video datoteka. Korišteni medij također može biti bilo koji popularni digitalni medij poput slike, videa ili teksta.

2.1. Steganografija

Steganografija slike jedan je od najpopularnijih i najpoznatijih oblika steganografije. Na slici 2.1 prikazana je klasifikacija tehnika steganografije slike.



Slika 2.1. Klasifikacija tehnika steganografije slike [2]

Kategorizacija se vrši prema vrsti korištene slike (2D ili 3D), procesu dohvaćanja (reverzibilan ili ne), domeni primjene (prostorna ili transformacijska domena), a izdvaja se i adaptivna steganografija. [3]

Tehnike prostorne domene izravno ili neizravno koriste razine intenziteta piksela slike za kodiranje bitova tajne poruke. Ove metodologije spadaju u neke od najjednostavnijih mehanizama u smislu složenosti ugrađivanja i dekodiranja. Primjenjuju umetanje bitova i manipulaciju šumom pomoću jednostavnih mehanizama. [4]

Vrijednosti domene transformacije također se mogu koristiti za integraciju tajnih bitova unutar slike. Kod ovih metoda, tajni bitovi se skrivaju ispod koeficijenata frekvencijskih podpojasa. Još jedna prednost je ta što je većina tehnika frekvencijske domene manje pogođena napadima kompresije, obrezivanja, skaliranja i rotacije. Zato su sustavi koji se temelje na transformaciji učinkovitiji u očuvanju kvalitete slike s ugrađenom porukom i čine ju manje uočljivom napadaču. [5]

Adaptivne metode poseban su slučaj prostornih i transformacijskih metoda. Adaptivna steganografija također se naziva „utemeljena na modelu“ (*eng. model-based*), „temeljeno na statistici" ili „maskiranje". Prilagodljivost se može uvesti u steganografske metode na više načina: odabirom ciljanih piksela u slici, prirodom modifikacije koju treba napraviti, brojem bitova ugrađenih u piksel itd. [6]

U ovom radu su implementirane dvije metode steganografije. Prva metoda je sekvencijalna LSB (*eng. least significant bit*) metoda, tj. metoda zamjene najmanje bitnoga bita. To je jednostavna metoda kojom se bitovi poruke spremaju zamjenom najmanje značajnih bitova vrijednosti piksela na slici s bitovima poruke. Kako ti bitovi imaju minimalan utjecaj na vizualni izgled slike, ova metoda omogućuje relativno besprijekorno ugrađivanje podataka. Druga metoda je patch LSB metoda. Ova metoda radi na istom principu kao prva, ali ne sprema bitove poruke u svaki piksel slike. Kod ove metode se najprije slika dijeli u određeni broj kvadratnih dijelova, zatim se računa standardna devijacija svakoga dijela zasebno i određuje broj znakova koji mogu biti spremljeni unutar njega. Svrha ove metode je smanjiti mogućnost detekcije postojanja poruka pomoću metoda steganaliza u zamjenu za smanjenje ukupne količine podataka koja se može spremiti unutar slike.

2.2. Steganaliza

Steganaliza je studija izvlačenja tajnih podataka iz stego-slika iz gledišta uljeza. To je složen proces budući da je slika sa skrivenim podacima često osigurana pomoću standarda šifriranja te može biti oštećena šumom ili bilo kakvim napadima. [7]

Postoje dva pristupa problemu steganalize. Jedan je osmisliti metodu steganalize koja je specifična za određeni steganografski algoritam. Drugi je razvoj tehnika koje su neovisne o steganografskom algoritmu koji se analizira. Svaki od ova dva pristupa ima svoje prednosti i nedostatke. Tehnika steganalize specifična za metodu steganografije dala bi vrlo dobre rezultate kada bi se testirala samo na toj metodi steganografije, a mogla bi biti neuspješna na svim drugim metodama. S druge strane, metoda steganalize koja je neovisna o algoritmu steganografije mogla bi općenito biti manje precizna, ali ipak dati prihvatljive rezultate na novim metodama steganografije. [8]

U ovom radu implementirana je metoda steganalize koja radi na temelju prepoznavanja statističkoga traga kojega steganografske metode ostavljaju prilikom ugrađivanja bitova poruke.

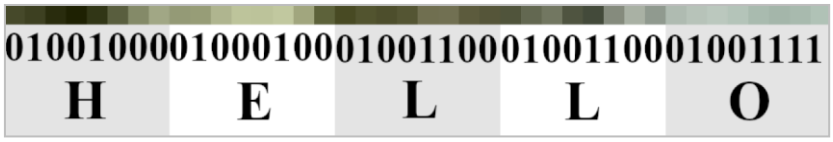
3. OPIS METODA

U ovome poglavlju opisane su dvije metode steganografije, sekvencijalna LSB metoda i patch LSB metoda. Nakon toga opisan je postupak dekodiranja poruka iz obrađenih slika istim steganografskim metodama. Na poslijetku opisana je metoda steganalize.

3.1. Sekvencijalna LSB metoda

Cilj sekvencijalne LSB metode je na jednostavan način sakriti poruku. Prolazeći redom kroz kanale boja unutar slike, svakom se pikselu na mjestu zadnjega, najmanje bitnog bita postavlja bit slova poruke. Slika se ovim načinom spremanja podataka mijenja minimalno. Ljudskom oku nisu vidljive ove promjene boje.

Za ovu metodu potrebno je imati samo sliku i poruku. Poruka se najprije pretvara u binarni oblik. Binarna vrijednost svakoga znaka poruke ASCII koda sprema se u jedan veliki niz znakova. Ovaj niz znakova se sastoji od znakova „0“ i „1“ te je 8 puta veći od poruke. Zatim se slijedno prolazi kroz sve piksele slike, počevši od prvoga, kroz 3 kanala slike: crveni, zeleni i plavi kanal. Svakom se pikselu postavlja vrijednost najmanje značajnog bita na vrijednost određenu znakovima poruke binarnog oblika. Na slici 3.1. nalazi se prikaz koji pobliže pojašnjava skrivanje poruke ovom metodom.



Slika 3.1. Skrivanje poruke „HELLO“ LSB metodom

Koraci sekvencijalne metode:

- Odabir slike i poruke
- Kopiranje slike
- Prevođenje poruke u binarni oblik (pretvaranje svakog ASCII znaka u 8 bitova)
- Inicijalizacija brojača ugrađenih bitova na vrijednost -1

- Prolazak kroz sve piksele slike, kroz sve kanale. Inkrementiranje brojača. Ako je brojač veći ili jednak od duljine poruke, ispisati poruku na zaslon o uspješnom spremanju binarne poruke u trenutku kada je brojač jednak duljini binarne poruke. Inače, ako brojač nije veći od duljine poruke postaviti vrijednost LSB bita na vrijednost koju sadrži binarna poruka na mjestu brojača.
- Provjera nakon prolaska kroz sve piksele slike: ako je brojač manji od broja bitova binarne poruke, ispisati pripadajuću poruku upozorenja (poruka nije u potpunosti skrivena unutar slike)
- Spremanje slike

3.2. Patch LSB metoda

Kao i kod sekvencijalne metode, patch LSB metoda također implementira steganografsku tehniku za skrivanje binarne poruke unutar najmanje značajnog bita, ali na malo kompliciraniji način. Prije spremanja, slika se dijeli na jednake dijelove određene visine i širine. Nakon toga se računa standardna devijacija svakoga dijela i određuje količina podataka koja smije biti spremljena u pojedini dio. Zatim se slijedno, počevši od prvoga dijela, sprema određeni broj znakova poruke. Potrebno je da spremljeni nizovi znakova svakoga dijela završavaju stop znakom kako bi znakovi poruke bili izdvojeni od znakova koji se dobiju dekodiranjem nepromijenjenih dijelova slike.

Odnos između standardne devijacije svakoga dijela i broja podataka koji se smiju spremiti određen je proizvoljno. Omogućavanjem spremanja više znakova za određene vrijednosti standardne devijacije smanjuje skrivenost poruke, tj. povećava se mogućnost otkrivanja postojanja poruke u slici. Na slici 3.2. prikazan je primjer spremljene poruke unutar slike patch LSB metodom. U ovome primjeru znak „$“ predstavlja stop znak.

He$	$	l$	$
2	0	1	0
$	lo $	Wo$	$
0	3	2	0
r$	l$	d!$	$
1	1	2	0
$	$	$	$
1	2	2	1

Slika 3.2. Primjer spremljene poruke „Hello World!“ unutar slike patch LSB metodom

Koraci patch metode:

- Odabir slike, poruke, dimenzije dijela i kanala slike
- Kopiranje slike
- Raspodjela slike na jednake, kvadratne dijelove odabrane dimenzije (pomoću biblioteke patchify [15])
- Računanje standardne devijacije svakoga dijela odabranog kanala
- Računanje količine podataka koja se smije spremiti u svaki pojedini dio slike odabranog kanala odvija se u 2 koraka. Rezultat prvog koraka dobije se prema formuli:

$$y = d^2 \cdot \frac{s' - 20}{60}$$

  gdje je d dimenzija dijela, a s' je vrijednost izračunate standardne devijacije unutar raspona $[26, 81]$ (za niže vrijednosti standardne devijacije od intervala se uzima vrijednost 26, a za više 81). Kao konačan rezultat se uzima vrijednost prvoga broja koji je manji od y te je djeljiv s brojem 8.
- Dodavanje stop znakova u poruku uzimajući u obzir izračunate količine podataka koje se smiju spremiti u dijelove slike. Ukoliko nije dopušteno spremiti niti jedan znak u određeni dio slike, svejedno dodati stop znak. Koraci dodavanja stop znakova:

    - Pretvoriti listu količina podataka u listu količine znakova (npr. listu *[0, 16, 8]* pretvoriti u listu *[0, 2, 1]*)
    - Oduzeti vrijednost 1 svakoj vrijednosti unutar liste osim vrijednosti 0.
    - Pretvoriti dobivenu listu u listu kumulativnih vrijednosti: svaku iduću vrijednost u listi zbrojiti s prethodnom, osim u slučaju prvog elementa.
    - Vrijednostima dobivene liste dodati vrijednost pozicije na kojoj se nalazi unutar liste
    - Za svaku vrijednost p unutar dobivene liste dodati stop znak unutar poruke na mjesto p
- Prevođenje poruke u binarni oblik (pretvaranje svakog ASCII znaka u 8 bitova)
- Inicijalizacija brojača na vrijednost 0
- Prolazak kroz svaki piksel svakoga dijela slike, postavljanje vrijednosti LSB bita na vrijednost koju sadrži binarna poruka na mjestu brojača. Svakih 8 ugrađenih bitova provjeriti čine li stop znak. Ako je stop znak ugrađen, nastaviti na idući dio slike. Uvećati brojač za broj ugrađenih bitova.
- Provjera nakon prolaska kroz sve piksele slike: ako je brojač manji od broja bitova binarne poruke, ispisati pripadajuću poruku upozorenja (poruka nije u potpunosti skrivena unutar slike)
- Spajanje dijelova natrag u sliku (pomoću biblioteke patchify [15])
- Spremanje slike

3.3. Dekodiranje

Spremanje podataka unutar slika nema smisla ukoliko ih nije moguće pročitati. Postupak dekodiranja poruka ovisi o postupku kodiranja. U slučaju sekvencijalne LSB metode, prvo je potrebno iščitati poruku iz piksela slike. Poruka je tada binarnog oblika. Nakon toga slijedi interpretacija te binarne poruke. Svakih 8 bitova binarne poruke se gleda kao jedno slovo jer su podaci spremljeni prema ASCII standardu.

Koraci dekodiranja sekvencijalne metode:

- Odabir slike

- Prolazak kroz piksele slike, spremanje LSB svakoga piksela u niz znakova
- Prevođenje dobivenog niza binarnih znakova u poruku (pretvaranje svakih 8 bitova u ASCII znak)
- Filtrirati znakove koji ne pripadaju poruci (odbaciti sve znakove poruke koji se nalaze nakon znaka čija se vrijednost ne nalazi između ASCII vrijednosti 32 i 127, uključujući)
- Spremanje poruke

Dekodiranje patch LSB metode započinje dijeljenjem slike u jednake dijelove koje je zatim potrebno dekodirati na isti način kao i kod sekvencijalne metode. Rezultati dekodiranja se na kraju spajaju u jedan niz znakova poruke. Broj dijelova u koje se slika dijeli mora biti isti kao i kod spremanja poruke na taj način.

Koraci dekodiranja patch metode:

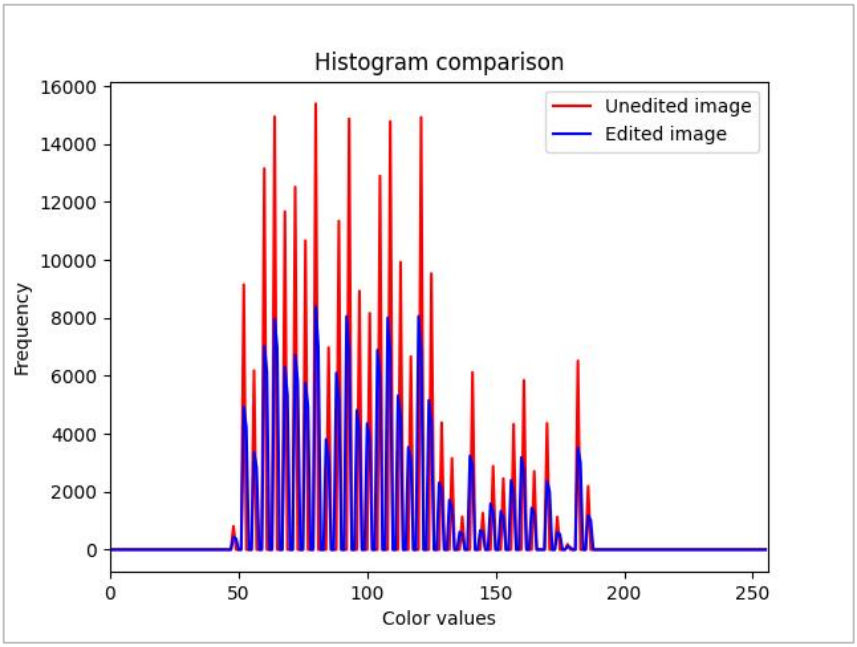
- Odabir slike, dimenzije i kanala
- Kopiranje slike
- Podjela slike na jednake, kvadratne dijelove odabrane dimenzije (pomoću biblioteke patchify [15])
- Inicijalizacija liste nizova znakova za spremanje rezultata dekodiranja pojedinih dijelova slike
- Prolazak kroz piksele svakoga dijela slike zasebno:
    - Spremanje LSB piksela u niz znakova
    - Prevođenje dobivenog niza binarnih znakova u poruku (pretvaranje svakih 8 bitova u ASCII znak)
    - Izbacivanje znakova iz poruke koji se nalaze nakon stop znaka
    - Dodavanje niza znakova u listu
- Spajanje nizova znakova u jedan niz
- Spremanje poruke

3.4. Metoda steganalize

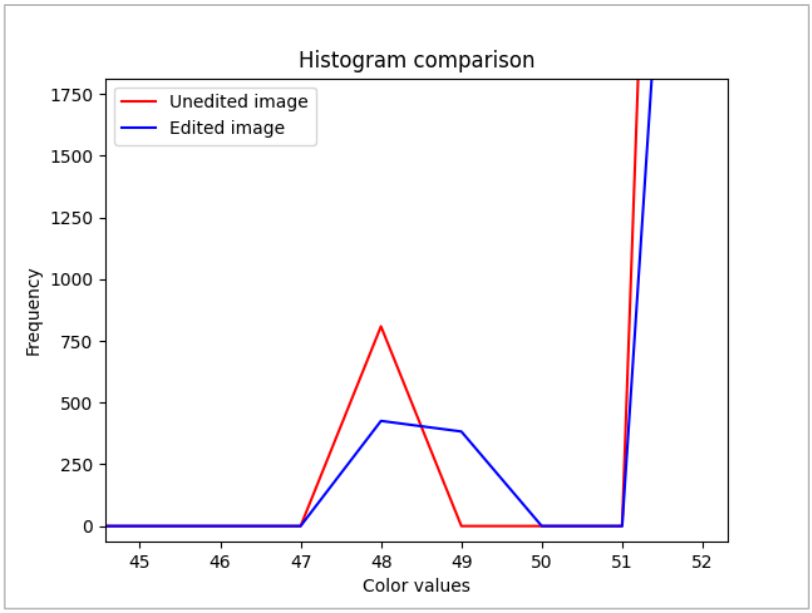
Jedna od jednostavnih metoda steganalize je metoda koja pomoću chi-square testa prepoznaje ima li slika ugrađene podatke. Povratna vrijednost ove metode je vjerojatnost postojanja poruke unutar slike.

3.4.1. Chi-square test

Chi-square je statistički test kojime je moguće prepoznati sličnost distribucija dvaju danih nizova brojeva. Ako je rezultat funkcije, odnosno vjerojatnost blizu 1, to znači da su vrijednosti prvoga niza vrlo slične vrijednostima drugoga niza. Kao prvi niz uzimaju se vrijednosti neparnih indeksa histograma kanala slike, tj. boje koja se analizira, a kao drugi niz uzimaju se aritmetičke sredine svaka 2 broja istoga histograma. Na ovaj način se prepoznaje da je slika nasumičnija nego što treba biti, to jest da su izmjenjene vrijednosti boja unutar slike. Na slici 3.3. prikazana je usporedba vrijednosti crvenog kanala slika, između slike bez poruke (crveno) i slike koja sadrži poruku (plavo). U ovome slučaju je cijeli kanal popunjen porukom te se zbog toga jasno vidi razlika distribucija frekvencija. Na slici 3.4. nalazi se približen prikaz razlike. Zbog načina na koji se poruka sprema unutar slike, vrijednosti porastu ili padnu za jedan te su zbog toga raspoređene bliže jedna drugoj, što omogućava detekciju pomoću chi-square funkcije.



Slika 3.3. Usporedba vrijednosti crvenog kanala slika



Slika 3.4. Približena usporedba histograma

Koraci metode steganalize (za 1 kanal slike):

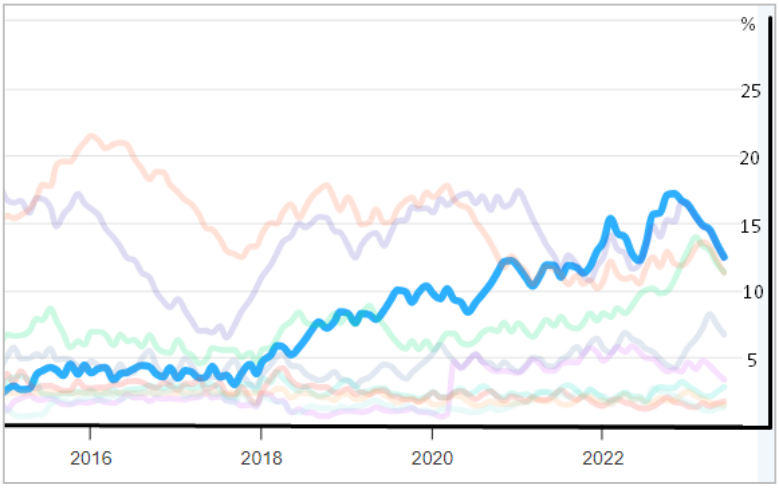
- Odabir slike i broja redova slike za analizu
- Izdvajanje broja redova iz slike
- Dohvaćanje histograma odabranih redova slike
- Računanje srednjih vrijednosti svaka 2 susjedna piksela i spremanje u prvu listu
- Kopiranje vrijednosti boje svakog piksela koji se nalazi na neparnim položajima u drugu listu
- Računanje chi-square testa s dvije liste
- Prikaz rezultata analize

4. IMPLEMENTACIJA RJEŠENJA

U ovom poglavlju biti će opisan korišteni programski jezik, metode steganografskog kodiranja i dekodiranja poruka te postupak steganalize.

4.1. Programski jezik python

Python je svestran programski jezik visoke razine poznat po svojoj jednostavnosti i čitljivosti, što ga čini popularnim i za početnike i za iskusne programere. Naglašava čitljivost koda i nudi opsežne biblioteke i programske okvire za razne aplikacije, uključujući web razvoj, analizu podataka i umjetnu inteligenciju. [9] Popularnost programskog jezika Python znatno je porasla zahvaljujući porastu u područjima znanosti o podacima i umjetne inteligencije. Na slici 4.1. prikazan je postotak korištenosti programskih jezika na kojemu je plavom bojom istaknut programski jezik Python. [10]



Slika 4.1. Prikaz grafa popularnosti Python programskog jezika u usporedbi s ostalima [10]

Za potrebe projekta ovoga rada korišteni su moduli: NumPy, OpenCV, Matplotlib, Scipy, Copy i patchify.

NumPy je temeljni paket za znanstveno računarstvo u pythonu. To je Python biblioteka koja pruža objekt višedimenzionalnog niza, razne izvedene objekte (kao što su maskirani nizovi i matrice) i asortiman rutina za brze operacije na nizovima, uključujući matematičke, logičke, manipulaciju oblikom, sortiranje, odabir, ulaz/izlaz, diskretne Fourierove transformacije, osnovna linearna algebra, osnovne statističke operacije, slučajna simulacija i još mnogo toga. [11]

Matplotlib je sveobuhvatna biblioteka za stvaranje statičnih, animiranih i interaktivnih vizualizacija u Pythonu. [12]

Scipy je softver otvorenog koda za matematiku, znanost i inženjerstvo. [13]

Copy je ugrađena Python biblioteka koja pruža generičke plitke i duboke operacije kopiranja. Naredbe dodjele vrijednosti varijablama u Pythonu ne kopiraju objekte, već stvaraju veze između varijable koju dodjeljujemo i varijable kojoj se dodjeljuje vrijednost. Za zbirke koje su promjenjive ili sadrže promjenjive stavke, ponekad je potrebna kopija tako da se može promijeniti jedna varijabla bez promjene druge. [14]

Patchify može podijeliti slike u male preklapajuće dijelove prema zadanoj veličini ćelije dijela i spojiti dijelove u originalnu sliku. [15]

4.2. Implementacija sekvencijalne LSB metode

Funkcija *GetEncodedImage* prikazana na slici 4.2. implementira steganografsku tehniku za skrivanje binarne poruke unutar najmanje značajnog bita svakog kanala boja slike. Kao ulazne parametre uzima sliku, poruku i izborni parametar *printEncodedMessage*.

```
def GetEncodedImage(self, image, message, printEncodedMessage = True):
    encodedImage = copy.deepcopy(image)
    stegUtils = SteganographyUtils.SteganographyUtils()
    binaryMessage = stegUtils.ConvertMessageToBinary(message)
    counter = -1
    for colorIndex in range(3):
        for i in range(np.size(encodedImage, 0)):
            for j in range(np.size(encodedImage, 1)):
                counter+=1
                if(counter >= len(binaryMessage)):
                    if(counter == len(binaryMessage) and printEncodedMessage == True):
                        self.PrintEncodedMessage(message, colorIndex)
                else:
                    if(binaryMessage[counter]=='1'):
                        a = encodedImage[i,j,colorIndex].astype(int)
                        a = a | 1
                        encodedImage[i,j,colorIndex] = a
                    else:
                        a = encodedImage[i,j,colorIndex].astype(int)
                        a = a & ~1
                        encodedImage[i,j,colorIndex] = a
    if(counter < len(binaryMessage)):
        print("Warning: Message is not fully encoded.")
    return encodedImage
```

Slika 4.2. Prikaz funkcije GetEncodedImage

Kopija izvorne slike stvara se pomoću metode *deepcopy()* python modula *copy* i pohranjuje u varijabli *encodedImage*. Ovime je osigurano da izvorna slika ne bude promijenjena tijekom procesa

kodiranja. Funkcija *getBinaryMessageString* vraća binarnu verziju dane poruke koju je moguće spremiti unutar piksela slike. Nakon dobavljanja binarne poruke slijedi prolazak kroz kanale slike (crveni, zeleni i plavi), počevši od prvoga piksela slike. Za svaki piksel kanala slike povećava se brojač koji služi za odabir idućeg bita binarne poruke koji se sprema. Brojač se nastavlja povećavati i kada sljedeći kanali slike budu na redu za spremanje binarne poruke. Ako je bit binarne poruke koji se treba spremiti '1', izvodi se bitovna operacija *ILI* na odgovarajućoj vrijednosti kanala boje u pikselu kako bi postavio najmanje značajan bit na 1. Ako je '0', izvodi se bitovna operacija *I* za postavljanje najmanje značajnog bita na 0. Ukoliko poruka nije u potpunosti spremljena, ispisuje se upozorenje. Naposlijetku se vraća varijabla *encodedImage* koja sadrži izvornu sliku s kodiranom porukom.

Način na koji je dobivena binarna poruka iz dane poruke prikazan je na slici 4.3. Funkcija *ConvertMessageToBinary* prolazi kroz svaki znak poruke i sprema ASCII vrijednost znaka u listu *characters*. Nakon što su sve vrijednosti spremljene, slijedi prolazak kroz svaku ASCII vrijednost znaka koja je čitana u binarnom formatu te spremljena u listu binarnih nizova znakova duljine 8 bita. Na samome kraju, funkcija vraća spojeni niz svih binarnih znakova.

```
def ConvertMessageToBinary(self, message):
    stringList = []
    characters = []
    for i in message:
        x=ord(i)
        characters.append(x)
    for i in characters:
        s = "{:08b}".format(i)
        stringList.append(s)
    return ''.join(stringList)
```

Slika 4.3. Prikaz funkcije ConvertMessageToBinary

4.3. Implementacija patch LSB metode

Funkcija *GetEncodedImage*, koja se nalazi unutar objekta zaduženog za spremanje poruka po dijelovima slike, prikazana je na slici 4.4.

```
def GetEncodedImage(self, message, image, step, channel=0, addStopSigns = True, stopSign = '$'):
    utils = SteganographyUtils.SteganographyUtils()
    image_height, image_width, channel_count = image.shape
    patch_height, patch_width = step, step
    patch_shape = (patch_height, patch_width, channel_count)
    patches = pf.patchify(copy.deepcopy(image), patch_shape, step=step)
    dataAmounts = self.GetChannelDataAmounts(patches, step, channel)
    if(addStopSigns == True):
        message = utils.AddStopSignsToMessage(message, dataAmounts, stopSign=stopSign)
    binaryMessage = utils.ConvertMessageToBinary(message)

    currentBitCounter = 0
    encodedPatches = np.empty(patches.shape).astype(np.uint8)
    for i in range(patches.shape[0]):
        for j in range(patches.shape[1]):
            patch = patches[i, j, 0]
            encoded_patch, currentBitCounter = self.EncodeMessageIntoPatch(patch, channel, binaryMessage,
                dataAmounts[int(i*image_width/patch.shape[0]+j)], currentBitCounter, addStopSigns=addStopSigns)
            encodedPatches[i][j][0] = encoded_patch

    if(currentBitCounter < len(binaryMessage)):
        print("Warning: Message is not fully encoded.")

    encoded_output_height = image_height - (image_height - patch_height) % step
    encoded_output_width = image_width - (image_width - patch_width) % step
    encoded_output_shape = (encoded_output_height, encoded_output_width, channel_count)
    encoded_output_image = pf.unpatchify(encodedPatches, encoded_output_shape)
    return encoded_output_image
```

Slika 4.4. Prikaz funkcije GetEncodedImage za patch metodu steganografije

(Slika 4.4.) Ova funkcija kao parametre prima sliku, poruku i parametar *step* koji određuje dimenziju svakog dijela slike (*eng. patch*). Osim navedenih parametra prima i tri izborna parametra: *channel, addStopSigns* i *stopSign*. Duboka kopija ulazne slike se dijeli na N jednakih dijelova. Ovi dijelovi imaju vrijednosti visine i širine jednake ulaznom parametru *step*. Zatim se računa standardna devijacija svakoga dijela slike odabranog kanala te se određuje koliko će slova poruke biti spremljeno u pojedini dio slike odabranog kanala. Na slici 4.5. prikazane su dvije funkcije kojima se određuju koliko će to slova biti za dani podatak standardne devijacije.

```
def GetDataAmount(self, std_value, step):
    std_value = self.Clamp(std_value, 26.0, 76.0)
    output = (step ** 2) * (((std_value-20)/5)/12)
    return (output - output%8)

def GetChannelDataAmounts(self, patches, step, channel):
    dataAmounts = []
    for i in range(patches.shape[0]):
        for j in range(patches.shape[1]):
            patch = patches[i, j, 0]
            std = np.std(patch[:,:,channel])
            dataAmounts.append(self.GetDataAmount(std, step))
    return dataAmounts
```

Slika 4.5. Prikaz funkcija GetDataAmount i GetChannelDataAmount

Nakon određivanja koliko se slova smije spremiti unutar svakog pojeding dijela slike potrebno je dodati stop znakove ukoliko je varijabla *addStopSigns* postavljena na vrijednost *True*. Na slici 4.6 prikazana je funkcija zadužena za to. U novu listu *data* se dodaju cijelobrojne vrijednosti varijable *dataAmounts*. Zatim se umanjuju vrijednosti unutar liste koje su veće od nule. Nakon toga se lista pretvara u listu kumulativnih vrijednosti, tj. svaka iduća vrijednost unutar nove liste je zbroj s prethodnom, osim u slučaju prvog elementa. Vrijednostima dobivene liste dodaje se vrijednost pozicije na kojoj se nalazi. Na mjestima određenim vrijednostima unutar liste redom se dodaju stop znakovi.

```
def AddStopSignsToMessage(self, message, dataAmounts, stopSign="$"):
    data = []
    for i in range(len(dataAmounts)):
        x = int(int(dataAmounts[i])/8)
        data.append(x)
    for i in range(len(data)):
        if(data[i] - 1 >= 0):
            data[i] = data[i]-1
    data = np.cumsum(data)
    for i in range(len(data)):
        data[i] = data[i] + i
    for position in data:
        message = message[:position] + stopSign + message[position:]
    return message
```

Slika 4.6. Prikaz funkcije AddStopSignsToMessage

Prije spremanja poruke u sve dijelove slike je potrebno pretvoriti poruku u binarni oblik. Ovo se postiže na isti način kao i u postupku spremanja poruke sekvencijalnom LSB metodom.

```
def EncodeMessageIntoPatch(self, patch, colorIndex, binaryMessage, dataAmount, currentBitCounter, addStopSigns=False, stopSign='$'):
    encodedPatch = patch
    isFirstForLoopIteration, isDataAmountReached, isMessageFullyEncoded, isStopSignReached = True, False, False, False
    counter = currentBitCounter-1
    for i in range(np.size(encodedPatch, 0)):
        if(isDataAmountReached or isMessageFullyEncoded or isStopSignReached):
            break
        for j in range(np.size(encodedPatch, 1)):
            if(isDataAmountReached or isMessageFullyEncoded or isStopSignReached):
                break
            counter+=1
            newDataAmount = dataAmount
            if(newDataAmount == 0):
                newDataAmount = 8
            if(counter >= len(binaryMessage)):
                isMessageFullyEncoded = True
                break;
            elif(counter == currentBitCounter+newDataAmount):
                isDataAmountReached = True
                break;
            else:
                if(addStopSigns and isFirstForLoopIteration==False):
                    if(counter%8==0 and counter!=0):
                        character = self.GetAsciiCharacter(binaryMessage, counter)
                        if(character == stopSign):
                            isStopSignReached = True
                            break
                self.ChangeBit(binaryMessage, counter, encodedPatch, i, j, colorIndex)
                isFirstForLoopIteration=False
    if(isMessageFullyEncoded):
        self.PrintMessage("MESSAGE ENCODED SUCCESSFULLY")
    currentBitCounter = counter
    return encodedPatch, currentBitCounter
```

Slika 4.7. Prikaz funkcije EncodeMessageIntoPatch

(Slika 4.7.) Funkcija *EncodeMessageIntoPatch* kao parametre prima dio slike *patch*, indeks kanala boje slike *colorIndex*, binarnu verziju poruke, količinu podataka *dataAmout* koliko se smije spremiti u dio slike, brojač *currentBitCounter* koji služi kao pokazivač na mjesto znaka binarne poruke koji se idući treba spremiti i dva izborna parametra *addStopSigns* i *stopSign*. Povratne vrijednosti su obrađeni dio slike i brojač uvećan brojem ugrađenih bitova. Unutar funkcije prolazi se kroz piksele danog dijela slike i sprema onoliko znakova koliko je određeno parametrom *dataAmount*. Kada dođe do limita zadanog ovom varijablom, dođe do kraja poruke ili dođe do stop znaka (ukoliko je izborni parametar *addStopSigns* postavljen na vrijednost *true*), tada se prekida prolazak kroz sve petlje.

Ukoliko se dodaju stop znakovi, svakih 8 bitova se pomoću funkcije *GetAsciiCharacter*, prikazane na slici 4.8, provjerava je li znak koji se treba spremiti jednak stop znaku. Ako je, tada se nakon spremanja stop znaka prekida spremanje daljnje poruke. Funkcijom *ChangeBit* prikazanom na slici 4.9. mijenja se zadnji bit trenutnog piksela ukoliko je potrebno za dani znak. Također, tada se u sliku zbog dodavanja stop znakova sprema nešto manje znakova od broja određenog parametrom *dataAmount*.

```
def GetAsciiCharacter(self, binaryMessage, counter):
    x = binaryMessage[counter-8:counter]
    decimal_num = int(x, 2)  # Convert binary number to decimal
    character = chr(decimal_num)  # Convert decimal to ASCII character
    return character
```

Slika 4.8. Prikaz funkcije GetAsciiCharacter

```
def ChangeBit(self, binaryMessage, counter, encodedPatch, i, j, colorIndex):
    if(binaryMessage[counter]=='1'):
        a = encodedPatch[i,j,colorIndex].astype(int)
        a = a | 1
        encodedPatch[i,j,colorIndex] = a
    elif(binaryMessage[counter]=='0'):
        a = encodedPatch[i,j,colorIndex].astype(int)
        a = a & ~1
        encodedPatch[i,j,colorIndex] = a
    else:
        print("Undefined behavior... ",binaryMessage[counter])
```

Slika 4.9. Prikaz funkcije ChangeBit

4.4. Implementacija postupka dekodiranja

Na slici 4.10. prikazana je funkcija kojom je moguće pročitati skrivene poruke LSB metodom. Ulazni parametri funkcije *Decode* su slika i tri izborna parametra: *messageFilter*, *decodeUntilStopSign* i *stopSign*.

```
def Decode(self, image, messageFilter = False, decodeUntilStopSign=False, stopSign='$'):

    hiddenBinaryMessage = self.ImageToBinaryMessage(image)
    hiddenMessage = self.ConvertBinaryToMessage(hiddenBinaryMessage)

    if(decodeUntilStopSign==True):
        hiddenMessage = self.RemoveAfterStopSign(hiddenMessage, stopSign)

    if(messageFilter == True):
        hiddenMessage = self.FilterNonRegularCharacters(hiddenMessage)

    return hiddenMessage
```

Slika 4.10. Prikaz funkcije Decode

Prvi korak dekodiranja je izvući binarnu poruku iz slike. Za taj korak zadužena je funkcija prikazana na slici 4.11. Redom se čita iz svakog piksela najmanje značajni bit i dodaje na kraj binarne riječi.

```
def ImageToBinaryMessage(self, image):
    hiddenBinaryMessage = ""
    counter = -1
    for color in range(3):
        for i in range(np.size(image, 0)):
            for j in range(np.size(image, 1)):
                counter+=1
                num = (image[i][j])[color]%2
                num = num.item()
                hiddenBinaryMessage += str(num)
    return hiddenBinaryMessage
```

Slika 4.11. Prikaz funkcije ImageToBinaryMessage

Nakon što je dohvaćena binarna riječ iz slike, tada je idući korak pretvoriti binarnu riječ u niz ASCII znakova. Ovo je odrađeno funkcijom prikazanom na slici 4.12. Svakih osam znakova binarne riječi pretvara se u ASCII znak i dodaje na kraj niza znakova.

```
def ConvertBinaryToMessage(self, hiddenBinaryMessage):
    hiddenMessage = ""
    binaryChar = ""
    counter = -1
    for n in hiddenBinaryMessage:
        counter += 1
        if(counter == 7):
            binaryChar = binaryChar + n
            if(int(binaryChar, 2)>31):
                hiddenMessage += chr(int(binaryChar, 2))
            binaryChar=""
            counter = -1
        else:
            binaryChar = binaryChar + n
    return hiddenMessage
```

Slika 4.12. Prikaz funkcije ConvertBinaryToMessage

Sljedeća dva koraka su izborna. U prvom izbornom koraku, prikazanom na slici 4.13, traži se stop znak unutar iščitane poruke. Ako dođe do stop znaka, ignoriran je stop znak i svi znakovi nakon. Drugi izborni korak, prikazan na slici 4.14, filtrira sve znakove koji se ne nalaze unutar raspona ASCII vrijednosti od 32 do 127.

```
def RemoveAfterStopSign(self, hiddenMessage, stopSign):
    old = hiddenMessage
    hiddenMessage = ""
    for c in old:
        if(ord(c)==ord(stopSign)):
            break;
        else:
            hiddenMessage = hiddenMessage + c
    return hiddenMessage
```

Slika 4.13. Prikaz funkcije RemoveAfterStopSign

```
def FilterNonRegularCharacters(self):
    old = hiddenMessage
    hiddenMessage = ""
    for c in old:
        if(ord(c)<32 or ord(c)>127):
            break;
        else:
            hiddenMessage = hiddenMessage + c
    return hiddenMessage
```

Slika 4.14. Prikaz funkcije FilterNonRegularCharacters

Ovim postupkom moguće je uspješno dekodirati sliku koja ima poruku spremljenu sekvencijalnom LSB metodom, ali neće uspjeti u dekodiranju slike s ugrađenom porukom patch metodom. Na slici 4.15. prikazana je funkcija prilagođena za dekodiranje tih slika. Ova funkcija ima dva nova potrebna parametra: *channel* i *step*. Parametar *channel* određuje kanal iz kojega će se iščitati poruka, a parametar *step* mora biti iste vrijednosti kao istoimeni parametar prilikom ugrađivanja poruke unutar slike. Odrađuje se isti postupak dijeljenja slike na jednake dijelove kao i kod ugrađivanja. Zatim se svaki dio slike dekodira istim postupkom dekodiranja kao i kod dekodiranja sekvencijalno spremljene poruke nad danim kanalom slike. Dekodirane poruke spremaju se u listu. Nakon što je dovršeno dekodiranje svakog dijela slike, vraća se niz znakova nastao spajanjem liste svih dekodiranih poruka.

```
def Decode(self, image, channel, step, decodeUntilStopSign = False, stopSign='$'):
    stegDecoder = StegDecoder.StegDecoder()
    image_height, image_width, channel_count = image.shape
    patch_height, patch_width = step, step
    patch_shape = (patch_height, patch_width, channel_count)
    patches = pf.patchify(copy.deepcopy(image) , patch_shape, step=step)

    output_patches = np.empty(patches.shape).astype(np.uint8)
    output_messages = []
    for i in range(patches.shape[0]):
        for j in range(patches.shape[1]):
            patch = patches[i, j, 0]
            hiddenMessage = stegDecoder.DecodeSingleChannel(patch, channel,
                decodeUntilStopSign=decodeUntilStopSign, stopSign=stopSign)
            output_messages.append(hiddenMessage)
            output_patches[i, j, 0] = patch

    output_message = "".join(output_messages)
    return output_message
```

Slika 4.15. Prikaz funkcije Decode za patch metodu steganografije

Na slici 4.16. prikazana je funkcija za dekodiranje samo jednoga kanala slike.

```
def DecodeSingleChannel(self, image, channel, messageFilter = False, decodeUntilStopSign=False, stopSign='$'):

    hiddenBinaryMessage = self.ImageToBinaryMessageFromChannel(image, channel)
    hiddenMessage = self.ConvertBinaryToMessage(hiddenBinaryMessage)

    if(decodeUntilStopSign==True):
        hiddenMessage = self.RemoveAfterStopSign(hiddenMessage, stopSign)

    if(messageFilter == True):
        hiddenMessage = self.FilterNonRegularCharacters(hiddenMessage)

    return hiddenMessage
```

Slika 4.16. Prikaz funkcije DecodeSingleChannel

Na slici 4.17. prikazana je funkcija koja izvlači binarnu poruku iz jednoga kanala slike.

```
def ImageToBinaryMessageFromChannel(self, image, channel):
    hiddenBinaryMessage = ""
    counter = -1
    for i in range(np.size(image, 0)):
        for j in range(np.size(image, 1)):
            counter+=1
            num = (image[i][j])[channel]%2
            num = num.item()
            hiddenBinaryMessage += str(num)
    return hiddenBinaryMessage
```

Slika 4.17. Prikaz funkcije ImageToBinaryMessageFromChannel

4.5. Implementacija metode steganalize

Funkcija prikazana na slici 4.18. prima dva parametra. Prvi parametar je slika za provjeru, a drugi parametar, *nRowsToCheck*, predstavlja broj redova unutar slike koji će se provjeravati. Vraća vjerojatnosti postojanja poruke unutar slike za svaki kanal. Rezultati se dobiju *chi-square* testom. Implementacija ovoga testa nalazi se na slici 4.19.

```
def Analyze(self, image, nRowsToCheck):
    if(nRowsToCheck>image.shape[1]):
        nRowsToCheck = image.shape[1]
    print(" Analyzing", nRowsToCheck, "rows (", nRowsToCheck/image.shape[1]*100, "%)")
    imagePartToCheck = image[0:nRowsToCheck,:,:]

    e_R, oddIndexElements_R = self.GetImageChannelFeatures(imagePartToCheck, 0)
    resultR = self.ChiSquareTest(oddIndexElements_R, e_R)
    e_G, oddIndexElements_G = self.GetImageChannelFeatures(imagePartToCheck, 1)
    resultG = self.ChiSquareTest(oddIndexElements_G, e_G)
    e_B, oddIndexElements_B = self.GetImageChannelFeatures(imagePartToCheck, 2)
    resultB = self.ChiSquareTest(oddIndexElements_B, e_B)

    return resultR, resultG, resultB
```

Slika 4.18. Prikaz funkcije Analyze

```
def ChiSquareTest(self, observed, expected):
    obs_sum, exp_sum, obsValues, expValues = self.GetSumsAndValues(observed, expected)
    x = (expValues/exp_sum)*obs_sum
    chi, p = sp.stats.chisquare(obsValues, x, 0)
    return p
```

Slika 4.19. Prikaz funkcije ChiSquareTest

Radi ispravnosti rezultata korištene funkcije *chisquare* biblioteke *scipy* [13], funkcijom *GetSumsAndValues* prije pozivanja *chisquare* funkcije su iz nizova izbačene vrijednosti na indeksima gdje se u listi *expected* nalaze nule kako ne bi došlo do dijeljenja s nulom. Osim toga, sume dvaju danih nizova vrijednosti moraju biti jednake, što se postiže prema funkciji

$$\{a_i\} = \frac{b_i}{\sum b} \sum a$$

gdje je a_i i-ti element prvog niza a, a b_i je element drugog niza b. Ovom funkcijom se vrijednosti skaliraju unutar niza, ali na način da ne utječu na rezultat chi-square funkcije. Način na koji se dobiju vrijednosti neparnih indeksa histograma kanala slike i aritmetičke sredine svaka 2 broja istoga histograma prikazan je na slici 4.20. Najprije se uzimaju podaci histograma dane boje, a nakon toga se iz tih podataka kreiraju dvije liste podataka. Prva lista sadrži aritmetičke sredine parova vrijednosti, a druga sve elemente neparnih indeksa.

```
def GetImageChannelFeatures(self, image, colorId):
    if(colorId < 0 or colorId > 2):
        colorId = 0
    histogramData, bin_edges = np.histogram(image[:, :, colorId], bins=256, range=(0, 256))
    averagedPairs = [(histogramData[i] + histogramData[i+1]) / 2 for i in range(0, len(histogramData)-1, 2)]
    oddIndexElements = histogramData[1::2]
    return averagedPairs, oddIndexElements
```

Slika 4.20. Prikaz funkcije GetImageChannelFeatures

5. REZULTATI I USPOREDBA STEGANOGRAFSKIH METODA

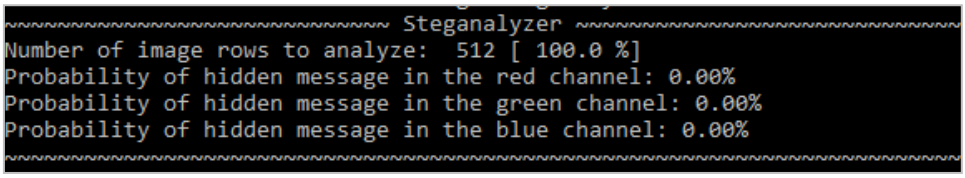
U ovome poglavlju prikazani su rezultati te su istaknute razlike između implementiranih steganografskih metoda. Kao primjere slika korištene su slike *lenna* [16] i *beach* [17], prikazane na slici 5.1.

Slika 5.1. (lijevo) „lenna.png“ (512x512) [16], (desno) „beach.png“ (512x512) [17]

Kao poruku za skrivanje unutar slike korišten je tekst *lorem ipsum* [18]: „*Lorem ipsum dolor sit amet, (...) tempor egestas arcu”*.

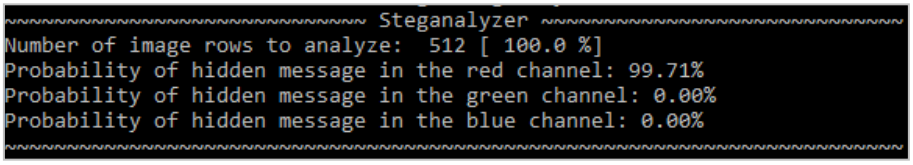
5.1. Rezultati sekvencijalne LSB metode

Rezultati steganalize nad originalnom slikom, bez izmjenjenih podataka unutar slike prikazani su na slici 5.2. Vjerojatnost iznosi 0% za svaki kanal, što je i očekivani rezultat.



Slika 5.2. Rezultat steganalize lenna.png bez skrivene poruke

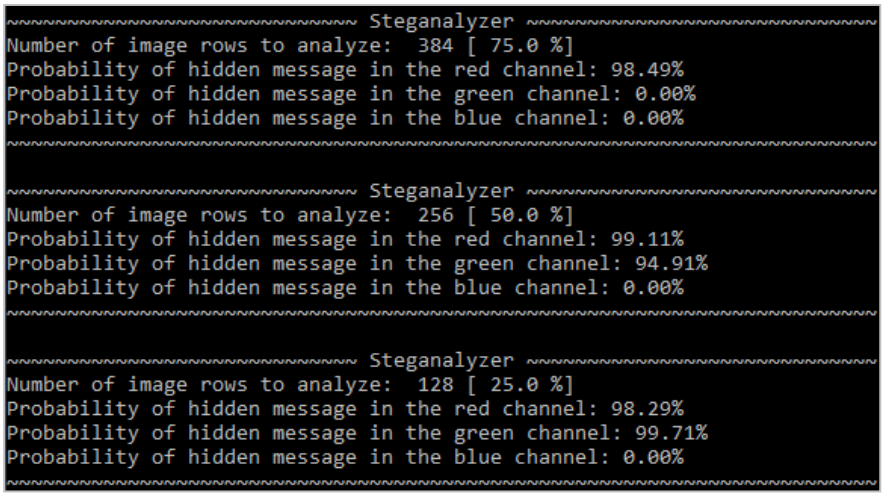
Rezultati steganalize slike koja sadrži poruku duljine 16 spojenih tekstova *lorem ipsum* prikazani su na slici 5.3. U ovome slučaju je cijeli crveni kanal slike izmjenjen. Algoritam prepoznaje postojanje poruke unutar crvenog kanala slike s visokih 99.71% sigurnosti.



Slika 5.3. Rezultat steganalize lenna.png s duljinom poruke većom od kanala slike

Spremljena poruka veća je od crvenog kanala. Dio poruke koji nije stao u crveni kanal se nalazi unutar zelenog kanala, no rezultat za zeleni kanal iznosi 0%. Ovdje je vidljivo da algoritam ne prepoznaje male poruke unutar velike slike.

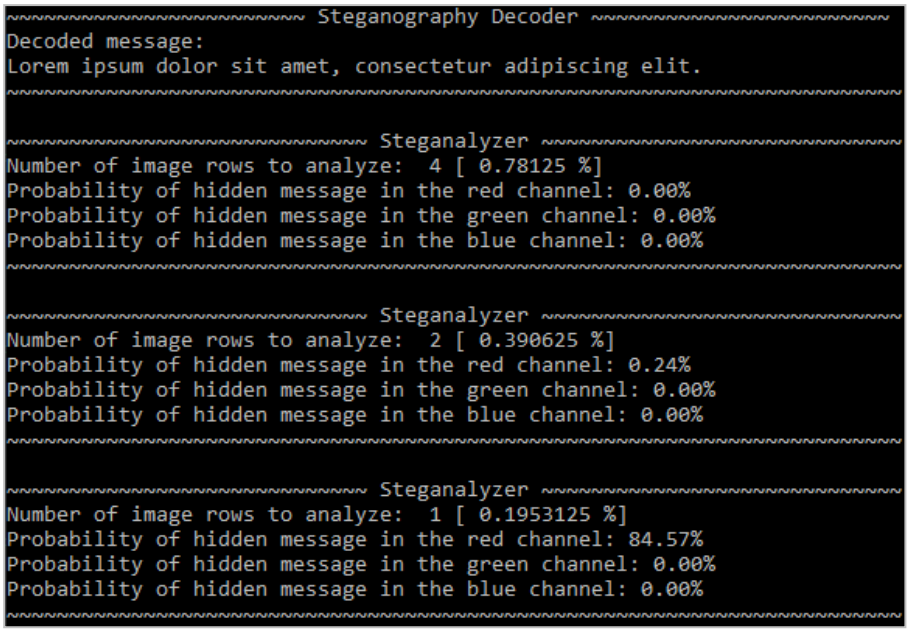
Kako bi se prepoznala manja poruka, potrebno je smanjiti sliku prije analize. Na slici 5.4. prikazani su rezultati triju analiza slike. Prva analiza uzima vertikalno smanjenu sliku za 25%. Svaka iduća analiza uzima smanjenu sliku vertikalno za još dodatnih 25%.



Slika 5.4. Rezultati steganalize lenna.png vertikalno smanjene slike

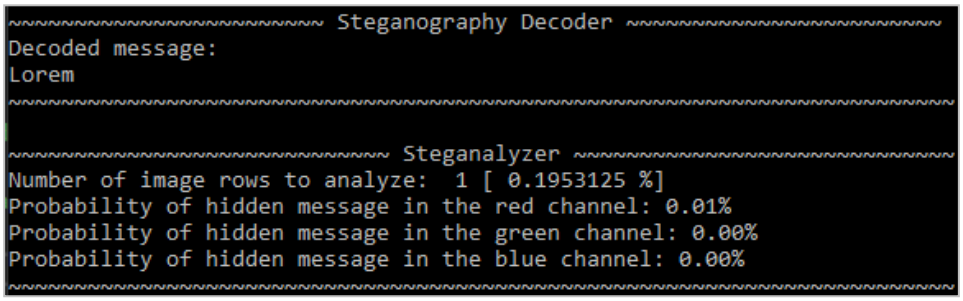
Kao u slučaju kada se provjeravala cijela slika, provjera 75% slike ne prepoznaje poruku unutar zelenog kanala. Kada je uzeto 50% ili manje, jasno se prepoznaje da postoji poruka.

Na slici 5.5 prikazani su rezultati steganaliza prvih redova slike kada je spremljena kratka poruka. Tek kada se provjerava samo jedan red se sa sigurnošću prepoznaje poruka.



Slika 5.5. Rezultati steganalize lenna.png kratke poruke

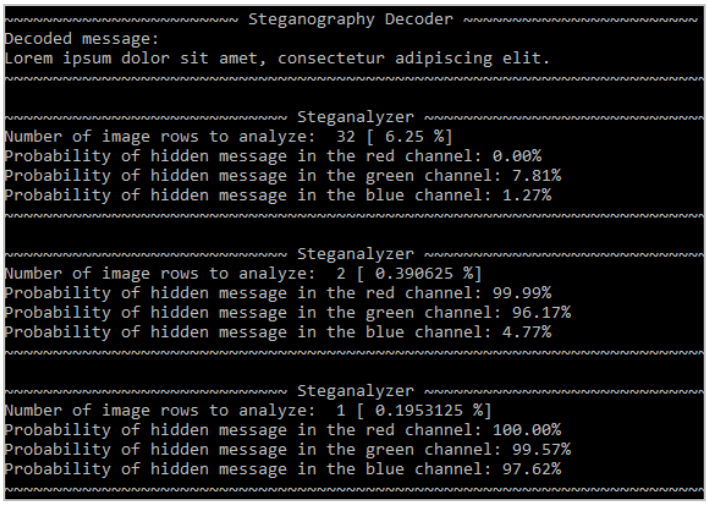
Što manju poruku slika sadrži, to ju je teže otkriti. Slika 5.6 prikazuje slučaj iznimno kratke poruke.



Slika 5.6. Rezultati steganalize lenna.png iznimno kratke poruke

Provjera samo jednoga reda daje vjerojatnost od vrlo niskih 0.01%, što je i dovoljno za zaključak da slika sadrži poruku, ali samo zato što su poznati rezultati nepromijenjene slike. Kada

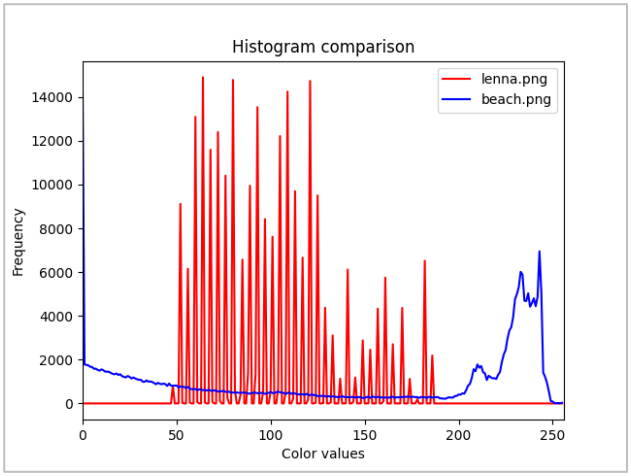
bi bio korišten algoritam izvan testnog okruženja, ne bi se moglo sa sigurnošću to zaključiti. Na slici 5.7. prikazani su rezultati steganalize slike s kratkom porukom unutar slike *beach* [17].



Slika 5.7. Rezultati steganalize beach.png kratke poruke

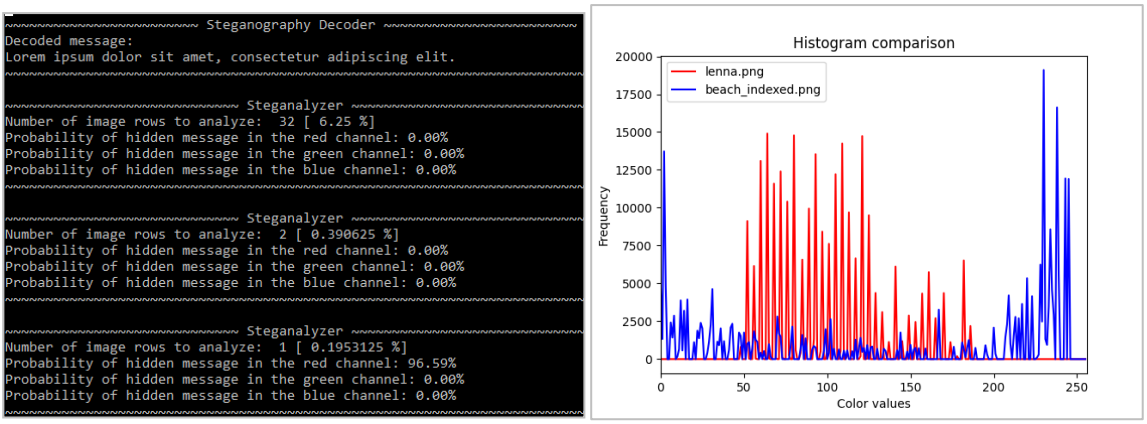
Prilikom analize smanjene slike *beach.png* pojavljaju se nekonzistencije koje daju lažno pozitivan rezultat. Postoci rezultata zelenog i plavog kanala su u nekim slučajevima blizu 100%, iako ne postoji skrivena poruka.

Smanjivanje slike nije problem u slučaju slike *lenna.png*, ali u slučaju *beach.png* je. Razlog zašto je *lenna.png* drukčija leži u frekvencijama boja. Na slici 5.8 prikazani su histogrami frekvencija nijansi crvene boje za dvije prethodno navedene slike.



Slika 5.8. Usporedba histograma slika lenna.png i beach.png

(Slika 5.8) Histogram slike *beach.png* izgleda kao kontinuirana funkcija, dok *lenna.png* više liči disretnoj. Što je slika bliža uniformnoj razdiobi, to je lakše prepoznati skrivene poruke unutar nje. Na slici 5.9 prikazani su rezultati steganalize slike *beach_indexed.png*, koja je promijenjena slika *beach.png* unutar aplikacije GIMP naredbom *indexed* (*image>mode>indexed*, *Maximum number of colors: 200*) [19]. Ovom funkcijom je promijenjen histogram slike smanjenjem broja nijansi boja pojedinog kanala, vidljivo na slici 5.10, usporedbi histograma *lenna.png* i *beach_indexed.png*.



Slika 5.9. (lijevo) Rezultati steganalize beach_indexed.png kratke poruke

Slika 5.10. (desno) Usporedba histograma slika lenna.png i beach_indexed.png

Slika 5.11. (lijevo) Prikaz razlike beach.png [17], (desno) beach_indexed.png

Na slikama 5.11. prikazana je razlika dviju slika plaže. Spremljena poruka će se teže prepoznati na desnoj slici nego na lijevoj.

5.2. Rezultati patch LSB metode

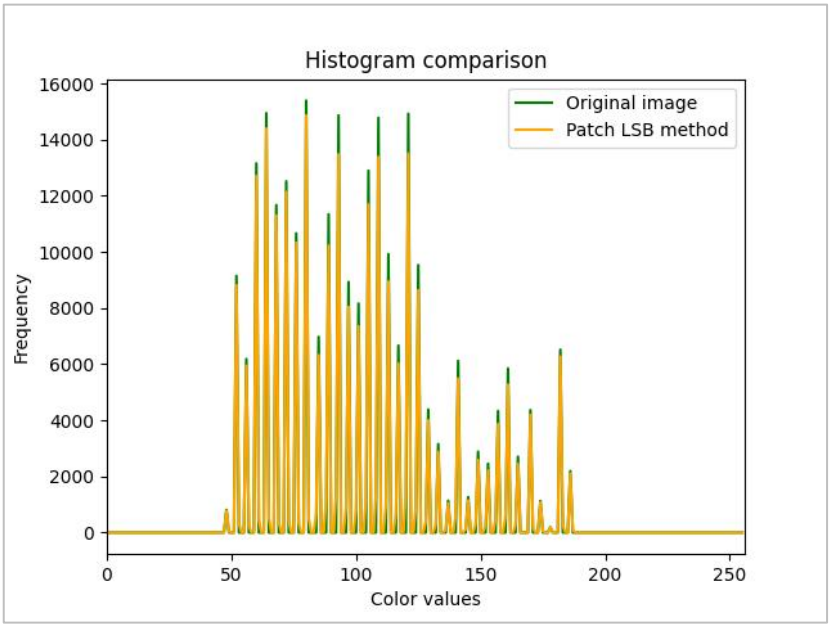
Na slici 5.12. prikazani su rezultati steganalize originalne slike i slike sa spremljenom porukom patch metodom. U oba slučaja rezultati prikazuju da nema skrivene poruke.

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Steganalyzer ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 -- Original image
Analyzing 512 rows ( 100.0 %)
Probability of hidden message in the red channel: 0.00%
Probability of hidden message in the green channel: 0.00%
Probability of hidden message in the blue channel: 0.00%
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

 -- Patch encoding (8x8)
Analyzing 512 rows ( 100.0 %)
Probability of hidden message in the red channel: 0.00%
Probability of hidden message in the green channel: 0.00%
Probability of hidden message in the blue channel: 0.00%
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Slika 5.12. Rezultati steganalize slike lenna.png bez poruke i spremljene poruke patch metodom

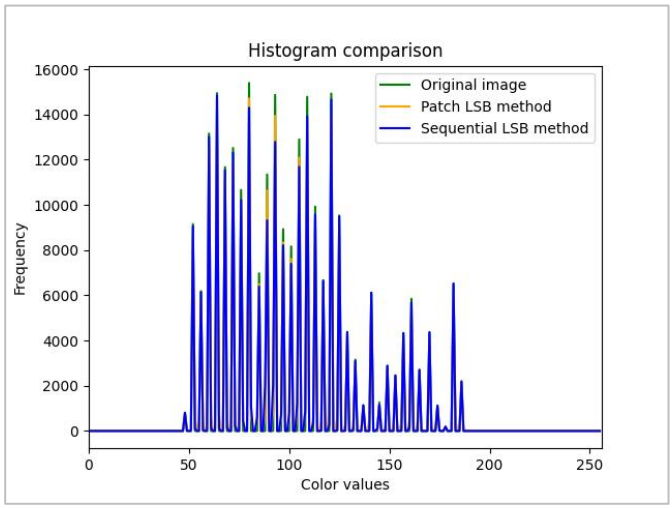
Na slici 5.13 prikazan je graf usporedbe histograma originalne slike i slike sa spremljenom porukom patch metodom.



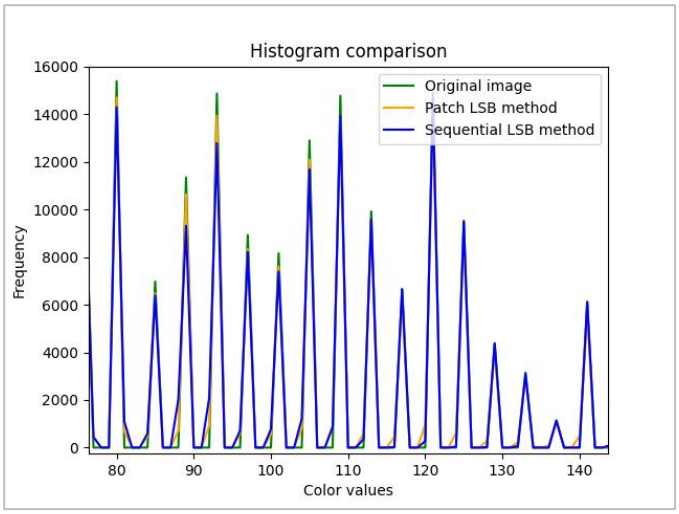
Slika 5.13. Usporedba histograma originalne slike lenna.png i spremljene poruke patch metodom

5.3. Usporedba sekvencijalne i patch LSB metode

Na slici 5.14. prikazana je usporedba histograma crvenog kanala slike *lenna* originalne slike i dviju implementiranih metoda steganografije, kod kojih je spremljena poruka duljine kojom se maksimalno iskorištava mogućnost spremanja patch metode. Na slici 5.15. uvećano je prikazan jedan dio ovoga grafa. Na grafovima je vidljivo kako histogrami slika kada je poruka skrivena odstupaju od histograma bez skrivene poruke. Također je moguće primjetiti da kod određenih vrijednosti sekvencijalna metoda puno više odstupa od patch metode.



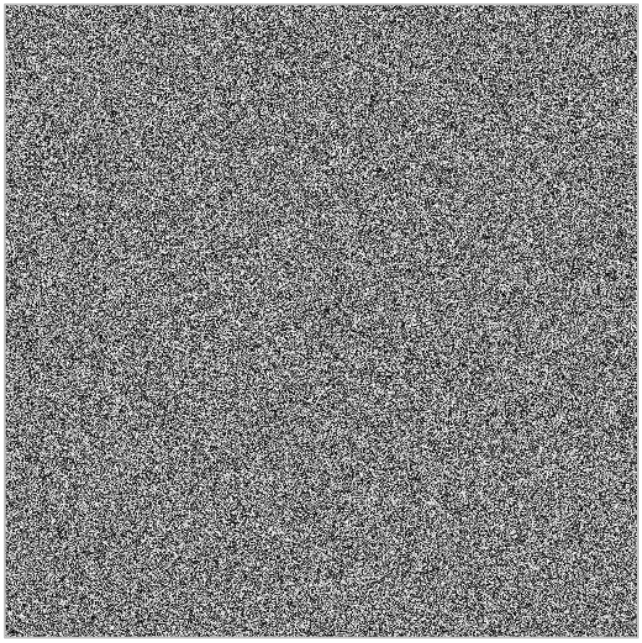
Slika 5.14. Prikaz usporedbe histograma originalne slike i dviju implementiranih metoda steganografije



Slika 5.15. Uvećani dio grafa prikazanog na slici 5.14.

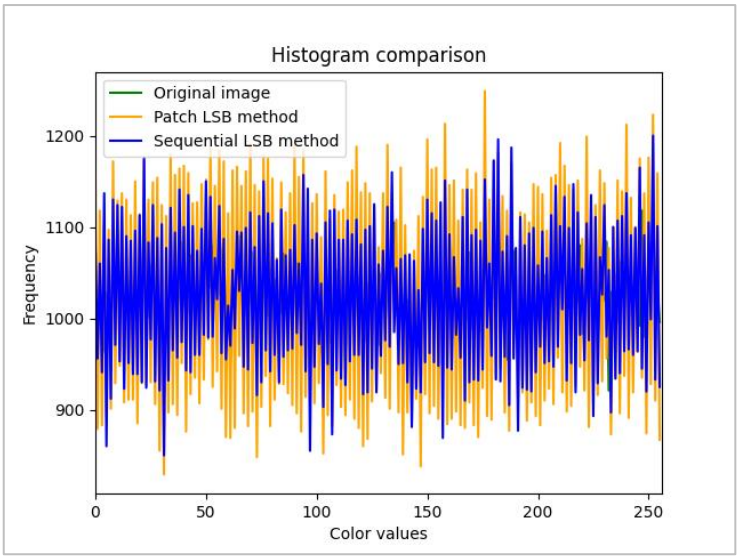
5.3.1. Usporedba po količini spremanja podataka

Mijenjanjem samo najmanje bitnih bitova piksela, unutar dane testne slike visine i širine 512 piksela s 3 kanala boje stane 786432 bita informacije, što znači da stane 98304 ASCII znakova ili 32768 ASCII znakova po kanalu slike. Sekvencijalnom metodom moguće je maksimalno iskoristiti ovaj broj podataka. Količina podataka koju patch metoda može spremiti je uvijek manja te varira s obzirom na statističke podatke slike, tj. o standardnoj devijaciji koja opisuje raspoređenost vrijednosti piksela slike. Više podataka može biti spremljeno na slikama s dobro raspršenim vrijednostima piksela nego na slikama s malo boja. Na slici 5.16. prikazana je slika *noise* unutar koje patch metoda može spremiti 22767 znakova unutar crvenog kanala slike, dok ih se unutar crvenog kanala slike *beach* može spremiti 1716, a crvenog kanala slike *lenna* samo 184 znakova. Navedene količine znakova se odnose na slučaj kada je uzeta dimenzija svakoga dijela od 8 piksela. Korištenjem druge dimenzije za dijelove može promijeniti količinu podataka koju metoda može spremiti.



Slika 5.16. Slika noise.png

Na slici 5.17. prikazana je usporedba histograma originalne slike *noise* i onih s ugrađenom porukom metodama steganografije.



Slika 5.17. Usporedba histograma slike noise

Problem kod spremanja unutar slike *noise* je u tome što je implementiranim postupkom steganalize uvijek detektirana poruka, bez obzira nalazi li se zaista poruka unutar nje. Na slici 5.18. prikazani su rezultati steganalize nad ovom slikom.

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Steganalyzer ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
 -- Original image
Analyzing 512 rows ( 100.0 %)
Probability of hidden message in the red channel: 100.00%
Probability of hidden message in the green channel: 100.00%
Probability of hidden message in the blue channel: 100.00%
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

 -- Sequential LSB method
Analyzing 512 rows ( 100.0 %)
Probability of hidden message in the red channel: 99.99%
Probability of hidden message in the green channel: 100.00%
Probability of hidden message in the blue channel: 100.00%
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

 -- Patch method (8x8)
Analyzing 512 rows ( 100.0 %)
Probability of hidden message in the red channel: 100.00%
Probability of hidden message in the green channel: 100.00%
Probability of hidden message in the blue channel: 100.00%
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Slika 5.18. Rezultati steganalize slike noise.png

Važno je istaknuti da se patch metodom sprema različiti broj znakova unutar drugih kanala slike jer svaki kanal ima svoj rezultat standardne devijacije.

6. ZAKLJUČAK

Steganografija je dobar način skrivanja informacija na očitom mjestu, barem dok potencijalna treća strana ne nasluti da bi mogla postojati skrivena poruka unutar slike. Ako posumnja da bi mogla postojati, moguće ju je detektirati postupkom steganalize jer svaka metoda ostavlja barem mali statistički trag. Na razinu neotkrivljivosti kod steganografije utječu razni čimbenici kao što su tehnika skrivanja, količina podataka koja se skriva, izbor slike, tehnika metode steganalize, ljudska percepcija itd. Napredak tehnologija i algoritama će vremenom utjecati na neotkrivljivost metoda.

U ovome radu implementirane su dvije metode steganografije slike i metode za dekodiranje poruka iz njih te metoda steganalize. Jedna metoda je jednostavna metoda koja znatno mijenja statistički otisak slike pa se postupkom steganalize lako prepoznaje postojanje spremljene poruke. Drugom metodom nastoji se poboljšati prvu. Smanjenjem statističkog otiska znatno se otežava prepoznavanje postojanja poruke steganalizom, ali u isto vrijeme smanjuje se maksimalna moguća količina podataka koja se može spremiti unutar slike. Metode dekodiranja obrnutim postupcima vraćaju poruku spremljenu unutar slike. Metodom steganalize, pomoću chi-square testa prepoznaju se slike unutar kojih se nalaze spremljene poruke.

Jedno od mogućih poboljšanja implementiranih steganografskih metoda bi bilo smanjenje potrebe bitova za pojedini znak. Na primjer, kada bi bili korišteni samo znakovi engleske abecede s brojevima, tada bi bilo dovoljno koristiti samo 6 bitova po znaku. Ovime bi se smanjila potreba za promjenom bitova unutar slike uz istu količinu spremljenih informacija. Kod patch metode bi se mogla povećati količina informacija koja se smije spremiti promjenom formule za izračun količine informacija pojedinog dijela. Budući da slike imaju različite rasporede frekvencija boja, nije moguće odrediti „najbolju“ formulu, ali ju se promatranjem rezultata steganalize nad različitim slikama sa spremljenim porukama ovom metodom može poboljšati. Glavni nedostatak ove metode je potreba za spremanjem stop znaka u svaki od dijelova slike. Zbog ovoga je lakše moguće prepoznati poruku promatranjem dijelova slike čije su boje piksela bliske. Poruke spremljene ovom metodom bez korištenja stop znakova potrebno je odvajati od nasumičnih znakova generiranih iz nepromjenjenih bitova.

LITERATURA

[1] M. Kharrazi, H. T. Sencar, N. Memon, Image Steganography: Concepts and Practice, Odjel za računarstvo i informatiku, Politehničko sveučilište, Brooklyn, NY 11201, SAD, str. 1, travanj 2004, dostupno na: http://sharif.edu/~kharrazi/pubs/ims04.pdf [Siječanj 2023]

[2] I. J. Kadhim, P. Premaratne, P. J. Vial, B. Halloran, Comprehensive survey of image steganography, Neurocomputing (299-326), str. 304, ožujak 2019, dostupno na: https://www.sciencedirect.com/science/article/pii/S0925231218312591 [Siječanj 2023]

[3] I. J. Kadhim, P. Premaratne, P. J. Vial, B. Halloran, Comprehensive survey of image steganography, Neurocomputing (299-326), str. 303, ožujak 2019, dostupno na: https://www.sciencedirect.com/science/article/pii/S0925231218312591 [Siječanj 2023]

[4] I. J. Kadhim, P. Premaratne, P. J. Vial, B. Halloran, Comprehensive survey of image steganography, Neurocomputing (299-326), str. 306, ožujak 2019, dostupno na: https://www.sciencedirect.com/science/article/pii/S0925231218312591 [Siječanj 2023]

[5] I. J. Kadhim, P. Premaratne, P. J. Vial, B. Halloran, Comprehensive survey of image steganography, Neurocomputing (299-326), str. 309, ožujak 2019, dostupno na: https://www.sciencedirect.com/science/article/pii/S0925231218312591 [Siječanj 2023]

[6] I. J. Kadhim, P. Premaratne, P. J. Vial, B. Halloran, Comprehensive survey of image steganography, Neurocomputing (299-326), str. 313, ožujak 2019, dostupno na: https://www.sciencedirect.com/science/article/pii/S0925231218312591 [Siječanj 2023]

[7] I. J. Kadhim, P. Premaratne, P. J. Vial, B. Halloran, Comprehensive survey of image steganography, Neurocomputing (299-326), str. 317, ožujak 2019, dostupno na: https://www.sciencedirect.com/science/article/pii/S0925231218312591 [Siječanj 2023]

[8] M. Kharrazi, H. T. Sencar, N. Memon, Image Steganography: Concepts and Practice, Odjel za računarstvo i informatiku, Politehničko sveučilište, Brooklyn, NY 11201, SAD, str. 14, travanj 2004, dostupno na: http://sharif.edu/~kharrazi/pubs/ims04.pdf [Siječanj 2023]

[9] Službena web stranica Pythona, dostupno na: https://www.python.org/ [Lipanj 2023]

[10] Statistike o korištenju programskih jezika, dostupno na: https://www.tiobe.com/tiobe-index/ [Lipanj 2023]

[11] NumPy biblioteka, dostupno na: https://numpy.org/doc/stable/ [Lipanj 2023]

[12] Matplotlib biblioteka, dostupno na: https://matplotlib.org/3.7.1/index.html [Lipanj 2023]

[13] Scipy biblioteka, dostupno na: https://docs.scipy.org/doc/scipy/ [Lipanj 2023]

[14] Copy biblioteka, dostupno na: https://docs.python.org/3/library/copy.html [Lipanj 2023]

[15] Patchify biblioteka, dostupno na: https://github.com/dovahcrow/patchify.py/blob/master/README.md [Lipanj 2023]

[16] Slika „lenna“, dostupno na: https://www.cosy.sbg.ac.at/~pmeerw/Watermarking/lena.html [Lipanj 2023]

[17] Slika iz koje je izrađena slika „beach“, dostupno na: https://unsplash.com/photos/n7DY58YFg9E [Lipanj 2023]

[18] Lorem ipsum, https://www.lipsum.com/feed/html

[19] GIMP indexed mode, dostupno na: https://docs.gimp.org/en/gimp-image-convert-indexed.html [Lipanj 2023]

SAŽETAK

Internetske tehnologije značajno su povećale količinu i dostupnost informacija. Nastali su novi izazovi u zaštiti podataka i osjetljivih informacija. U ovome radu predstavljene su različite metode steganografije za skrivanje poruka unutar digitalnih slika, kao i metoda steganalize kojom je moguće otkriti samu postojanost poruka. Implementirane su dvije metode steganografije i metoda steganalize unutar Python okruženja, gdje prva metoda radi na principu spremanja podataka u najmanje značajne bitove piksela slike, dok druga metoda steganografije radi na istom principu, ali uz dodatna ograničenja radi smanjenja mogućnosti otkrivanja postojanosti poruke unutar slike.

Ključne riječi: steganografija, steganaliza, računalne slike, Python

ABSTRACT

Internet technologies have significantly increased the amount and availability of information. New challenges have arisen in the protection of data and sensitive information. This paper presents different methods of steganography for hiding messages within digital images, as well as a steganalysis method that can detect the very presence of messages. Two methods of steganography and a method of steganalysis were implemented within the Python environment, where the first method works on the principle of saving the data at least significant bits of image pixels, while the second method of steganography works on the same principle, but with additional restrictions to reduce the possibility of detecting the persistence of a message within an image.

Keywords: steganography, steganalysis, computer images, Python

ŽIVOTOPIS

Toni Kunštek rođen je 6.10.1999. godine u Osijeku. Nakon završenog osnovnog obrazovanja koje je započelo 2006. godine u Osnovnoj školi Ivana Kukuljevića Belišće upisuje srednju školu. Obrazovanje nastavlja 2014. godine u Elektrotehničkoj i prometnoj školi Osijek gdje upisuje smjer tehničar za mehatroniku. 2018. godine upisuje preddiplomski studij, smjer računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku koji je u sklopu Sveučilišta Josipa Jurja Strossmayera. Nakon uspješnog završetka preddiplomskog studija upisuje diplomski studij računarstva, smjer Informacijske i podatkovne znanosti na istom fakultetu.

_____

Toni Kunštek

PRILOZI

[1] Github repozitorij: https://github.com/KunstekT/DiplomskiRad

[2] CD s elektroničkom verzijom diplomskog rada i projektnim zadatkom